The ICF3D Code

A. I. Shestakov M. K. Prasad J. L. Milovich
N. A. Gentile J. F. Painter G. Furnish
Z. Motteler T.-Y. B. Yang P. F. Dubois

Introduction

ICF3D is a three-dimensional radiation hydrodynamics simulation computer code being developed for ICF applications. It has a number of distinguishing features:

- Portable; works on uniprocessors and massively parallel processors (MPP).
- Written in the object-oriented programming (OOP) language C++.
- Based on unstructured grids.
- Discretized using finite elements; the hydrodynamics is modeled using discontinuous functions.

We believe these features are important for a variety of reasons.

These design codes must be portable, if they are to run on future computers. Computers are evolving at such a rapid pace that today's supercomputer will be obsolete in two to three years. However, software development is a painfully slow and labor-intensive task.[1] Thus, it is important that codes developed now can run efficiently on tomorrow's computers. Today's supercomputer is a parallel machine, a collection of individual "boxes" each with its own memory and with one or more processing elements (PEs). ICF3D is written to take advantage of this architecture. It parallelizes by decomposing physical space into nearly disjoint subdomains and relies on explicit calls to system message-passing routines. This approach allows us to scale the computation. If more boxes are available, bigger problems can be run.

In order to have reusable code, software should be robust and modular. This is facilitated by an OOP approach. In a modular code, if something needs to be rewritten or modified, one can retain the trusted components. Functions or modules often need to protect their internal variables from inadvertent corruption by other routines; such protection leads to "data hiding." For better organization, one may wish to use "classes," or to define new entities, each with their individual methods, e.g., a cell and a means of calculating its volume. These requirements are easily accommodated by OOP languages such as C++. Traditionally, scientific software has used FORTRAN. ICF3D is embracing C++. This approach is not without risks. Although OOP in general, and C++ in particular, is now widely used, it rarely appears in computational physics. In the past, C++ compilers were notoriously slow in optimizing code, and for scientists, speed is nearly as important as accuracy. This state of affairs is changing, and the pessimistic results previously reported by Haney[2] are no longer true.[3]

Codes based on unstructured grids can easily model real experiments with complicated geometries. For example, in an indirectly driven ICF experiment, a spherical capsule is embedded in a nearly vacuous cylindrical hohlraum with partially opened ends. Some experiments may have additional shields inside the hohlraum to protect the capsule. During the experiment, the walls and capsule undergo significant displacement. In modeling, a traditional, fully structured mesh will have difficulty simulating and resolving the initial configuration and its subsequent motion. An unstructured grid is useful as it allows different cell types to be connected. The extra overhead in allowing complicated cells is offset by the flexibility afforded when the original domain is discretized and/or the problem regridded.

Simulation codes of this type benefit by being discretized using finite elements. Unstructured grids and complicated geometries naturally lead to finite element (FE) methods. ICF3D's grid consists of a collection of hexahedra, prisms, pyramids, and/or tetrahedra. Processes such as diffusion are modeled by nodal FE methods in which the variables are given a continuous representation throughout the domain. The hydrodynamics is simulated by a novel scheme[4] based on the discontinuous FE method. This allows a natural representation of inherently discontinuous phenomena, such as shocks.

In this article, the first section provides an overview of ICF3D's modules, discusses how the physics packages are coupled, and describes the individual packages in more detail. The next section presents some results. In the conclusion, we describe our future plans.

We have a couple of clarifications to make about our word usage. In internal discussions, the name "ICF3D" sometimes denotes both the "stand-alone" physics code as well as the environment used to initialize problems, execute them, control the execution, and analyze the results. To avoid confusion, in this article ICF3D refers only to the stand-alone code. In addition, throughout the article, we use "module" and "package" interchangeably to denote a set of routines that perform a specific task.

ICF3D Modules

ICF3D runs on a variety of machines. Its I/O is in a special format, which is described in this article in "ICF3D Initialization"(see p. 168). Once the input files are prepared, ICF3D may be run like any other C++ program. ICF3D also has an interactive controlling environment. The interpretive language Python[5] controls the execution.

In the following subsections, we discuss the ICF3D modules. The modules consist of one or more C++ functions. When properly designed, modules should be easy to check, and if the need arises, easy to replace with better modules. The modules' execution is controlled by user-set parameters. Most modules can be run separately. This code has separate modules for Initialization, Hydrodynamics, Heat Conduction, Radiation Diffusion, Equation of State (EOS), and Parallel Processing.

One important issue is how to couple the physics packages. The problem is complicated since the packages may have different representations of the variables, e.g., cell or node centered. This difference is exemplified by the hydro and radiation diffusion packages. The former evolves equations for the density ρ, the momentum density $\rho\mathbf{v}$, and the total energy density ρE. The radiation diffusion package couples the temperature T to the spectral radiation energy density u_ν where ν is the photon frequency. The coupling difficulty arises because hydro variables have a discontinuous FE representation, whereas any quantity undergoing diffusion (a second-order differential operator) must have a continuous representation, if the diffusion is modeled by FE. A straightforward coupling of a nodal u_ν to a zonal T may create anomalous diffusion.[6] Hence, the radiation-to-matter coupling should be done with functions having similar (nodal and continuous) representations.

The equations of interest are the conservation laws for mass, momentum, and total matter energy, respectively:

$$\partial_t \rho + \quad \mathbf{F}_\rho = 0 \quad , \tag{1}$$

$$\partial_t(\rho\mathbf{v}) + \quad \mathbf{F}_{\rho\mathbf{v}} = \rho\mathbf{g} \quad , \tag{2}$$

and

$$\partial_t(\rho E) + \quad \mathbf{F}_{\rho E} = \rho\mathbf{g} \quad \mathbf{v} + H_\varepsilon + S_\varepsilon + \int K_{\nu\varepsilon} d\nu \quad . \tag{3}$$

Equation (3) is coupled to the transport (diffusion) equation of the radiation field

$$du_\nu / dt = \quad \left(D_\nu \quad u_\nu\right) - K_{\nu\varepsilon} \quad . \tag{4}$$

In Eqs. (1) to (3), $\mathbf{F}_i$ denotes the flux of i, i.e., $\mathbf{F}_\rho v_x = \rho v_x \mathbf{v} + p$, and $\mathbf{F}_{\rho E} = (\rho E + p)\mathbf{v}$, where v_x is the x velocity component and p is the pressure. In Eqs. (2) to (4), $\rho\mathbf{g}$ is an external force density, ε is the internal energy, H_ε is the heat conduction term, S_ε is a source of energy (e.g., due to laser deposition), $K_{\nu\varepsilon}$ describes the radiation-to-matter coupling, d/dt is the Lagrangian derivative, and D_ν is the diffusion coefficient of the radiation field. In the future, when ε is split into separate electron and ion components, $K_{\nu\varepsilon}$ will denote the radiation-to-electron coupling, and S_ε, if due to a laser, will be an electron source.

Equation (3) is solved by operator splitting. At the start of the time cycle, we compute all the coefficients we need, such as conductivity. Then, we do a hydro step; Eqs. (1) to (3) are advanced together except that in Eq. (3) the H, S, and K terms are ignored.

The hydro module allows for the passive advection of an arbitrary number of other variables. Presently, this feature is only used for the "mass fractions." In the future, Eq. (4) would also be solved by operator splitting. The convective part would be done by the hydro, while the transport and radiation-to-matter coupling would be done at the end of the cycle.

The hydro module produces an intermediate total energy $E^{(1)}$. We now introduce the subscript d to denote variables whose numerical representation is discontinuous. If the grid consists of only regular hexahedra, there are eight cells adjacent to each node. Hence, f_d denotes a function with eight values per node. At the conclusion of the hydro step, we compute an intermediate internal energy

$$\varepsilon_d^{(1)} = E_d^{(1)} - \upsilon_d^2/2 \quad . \tag{5}$$

This step is potentially dangerous since Eq. (5) implicitly assumes that $\upsilon_d^2/2$ is a valid representation for the kinetic energy e_k. Unfortunately, e_k is not computed directly but is only derived by squaring the velocity. The difficulty is illustrated by considering an ideal gas for which we require $\varepsilon \geq 0$. Since the code evolves ρ, $\rho\mathbf{v}$, and ρE, there are no explicit assurances that $E \geq \upsilon_d^2/2$.

The variables ρ_d and $\varepsilon_d^{(1)}$, and the EOS yield the pressure $p_d^{(1)}$ and the temperature $T_d^{(1)}$. The other physics packages compute changes to the internal energy $\Delta\varepsilon_d$. The cycle concludes by computing the final energies

$$\varepsilon_d = \varepsilon_d^{(1)} + \Delta\varepsilon_d \text{ and } E_d = E_d^{(1)} + \Delta\varepsilon_d \quad . \tag{6}$$

The change $\Delta\varepsilon_d$ consists of the H, S, and K terms that were ignored by the hydro step. At the end of the time cycle, we have both a continuous nodal T and a discontinuous ε_d. The two variables may not be consistent. If necessary, we use the EOS to get a self-consistent pressure p_d and a temperature T_d from ρ_d and ε_d. Since T_d has a discontinuous FE representation, it may not agree with the continuous FE (nodal) value obtained by coupling to the radiation. To be more precise, after multiplying through by the time step, we write

$$\rho\Delta\varepsilon = H_\varepsilon + S_\varepsilon + \int K_{\nu\varepsilon} d\nu \quad . \tag{7}$$

The energy difference is expressed as a temperature difference

$$\Delta\varepsilon = c_v\left(T - T^{(1)}\right) \quad , \tag{8}$$

where

$$c_v = \frac{\partial\varepsilon}{\partial T}\bigg|_{\rho^{(0)},T^{(0)}} \tag{9}$$

is the specific heat computed at the start of the time cycle.

We now specify how the variables of Eqs. (7) to (9) are defined. The continuous FE representation of the temperature at the end of the hydro at the j^{th} grid point is

$$T_j^{(1)} = \frac{\int_\Omega dV \ \varphi_j \ \rho_d \ c_{v,d} \ T_d^{(1)}}{\int_\Omega dV \ \varphi_j \ \rho_d \ c_{v,d}} \quad , \tag{10}$$

where Ω denotes the entire domain and φ_j is the FE basis function centered at the j^{th} grid point. The integrals in Eq. (10) are sums over all the cells that support φ_j. Furthermore, the integrals are lumped, i.e., for any

$$\int_\Omega dV \ \varphi_j \ f = \sum_{cell} \left(\int_{\Omega \cap cell j} dV \ \varphi_j \right) f_{d,j} \quad , \tag{11}$$

where $cell_j$ is a cell with x_j as one of its vertices and $f_{d,j}$ is the discontinuous value in $cell_j$ at $x = x_j$. In the FE discretization of the heat conduction and radiation-to-matter coupling equations, we lump all but the transport term. Each equation is discretized by multiplying by $\Delta t \ \varphi_j$ and integrating over Ω.

Once $T^{(1)}$ is known, we use operator splitting and FE to first do the heat conduction and the energy deposition

$$\rho c_v\left(T^{(2)} - T^{(1)}\right) = \quad \kappa^{(0)} \quad T^{(2)} + S\left(\rho, T^{(1)}\right) \quad . \tag{12}$$

In Eq. (12), the heat transport is implicit in $\Delta T^{(2)}$, but the conductivity is computed using values at the start of the cycle. The source S is explicit. If it is due to a laser, and its deposition depends on both ρ and T, we use the latest values, the ones obtained after the hydro step. For laser deposition, we need continuous representations of ρ and T in order to compute their gradients. A continuous (nodal) density is given by its nodal values

$$\rho_j = \frac{\int_\Omega dV \ \varphi_j \ \rho_d}{\int_\Omega dV \ \varphi_j} \quad , \tag{13}$$

where the integral in the numerator is lumped, as in Eq. (11). Using the nodal values, ρ is easily obtained from the analytic, continuous representation

$$\rho(x,y,z) = \sum_j \varphi_j(x,y,z) \ \rho_j \quad . \tag{14}$$

The gradient is continuous within cells and discontinuous across cell faces.

Equation (12) is followed by the radiation-to-matter coupling. Equation (4) is coupled to

$$\rho c_v\left(T^{(3)} - T^{(2)}\right) = \int K_{\nu\varepsilon}^{(3)} d\nu \quad , \tag{15}$$

where K is implicit in the final, continuous temperature. Equations (4) and (15) are solved by standard FE techniques; everything except the transport term in Eq. (4) is lumped. The result is a continuous representation for both the final radiation energy density u_ν and matter temperature $T^{(3)}$.

To summarize, T is a derived quantity and ε is fundamental. During the heat conduction and radiation-to-matter coupling, we keep track of the energy changes in each cell. If the changes are small, then the ε after the hydro is not modified much. Hence, any sharp features are not smeared out. In particular, if there are no matter energy sources, very small heat-transport coefficients, and insignificant radiation-to-matter coupling, then the hydro should work as if it is the only package running. Similarly, if we have only hydro and a zonal matter energy source running, and the source is distributed according to Eq. (11), then the results should still stay sharp.

ICF3D Initialization

The grid consists of three types of objects: cells, faces, and nodes. A node is characterized by its sequence number and its coordinates. Cells and faces use the object-oriented concept of inheritance. Faces may be quadrilateral or triangular, whereas four types of cells are allowed: tetrahedra, pyramids, prisms, and/or hexahedra. The cell, face, and node objects are related. For example, a hexahedron has 6 faces and 8 nodes. Each interior face has two cells on either side. The cells need not be regular; the quadrilateral faces need not be coplanar. However, the cells cannot be so distorted to preclude an isoparametric mapping to regular elements, e.g., quadrilaterals to a unit cube. For Lagrangian node motion, this implies that if the grid becomes sufficiently distorted, the problem must be regridded. We do not yet have a means of regridding.

We have not yet written a general, unstructured mesh generator. In order to run test problems, we have instead written a simple, separate mesh generator that outputs mesh description files for subsequent reading by ICF3D. However, we stress that ICF3D is written to run on completely unstructured meshes. Our generator is described in "Problem Generation" (see p. 172); it writes ICF3D input files that describe the mesh in the Advanced Visual Systems (AVS) Unstructured Cell Data (UCD) format. (AVS is a commercial software visualization product.) This format, which consists of two lists, is a terse description of the grid. The first is a node list; each node is described by its unique sequence number (an integer) and by the values of its coordinates (three real numbers). The second is a cell list; each cell is described by its type—e.g., a pyramid—and by a list of integer sequence numbers that comprise the cell's vertices. The cell's vertices must be given in a prescribed way, e.g., a pyramid's apex is listed first.

We have used the UCD because it is commercially available, but it is not adequate for use in generating an unstructured mesh. For example, the format does not explicitly list which cells share given a node. Also the format never mentions faces. To define the mesh, ICF3D requires that the different objects (cells, faces, and nodes) have a set of interconnecting pointers to describe which cells lie on either side of a face, which nodes make up the face, etc.

The discontinuous hydrodynamic scheme imposes even more requirements on the mesh description. The dependent variables are cell–node based. For example, each cell's density is described by its value on the vertices. Since the hydrodynamic variables are allowed to be discontinuous, a neighboring cell has different nodal values. Thus, such variables are considered "doubly indexed": once over cells, then again over the cell's nodes. When the hydrodynamic face fluxes are computed, the routine loops over each face, follows the pointer to each of the adjoining cells, and goes to the appropriate node to pick up the value. This requires the mesh to provide an additional set of pointers: face $\rightarrow$ cell $\rightarrow$ node.

Some physics packages are more efficient if additional connectivity information is supplied. For the hydrodynamics, the limiting routine, which deletes unphysical extrema, requires that each cell point to all other cells that share its nodes. The continuous, piecewise-linear, nodal FE scheme that discretizes the second-order elliptic operator imposes a similar requirement. Each node, which by construction is a vertex of one or more cells, needs a list of the other vertices. On a regular hexahedral grid, the nodal neighbor list gives rise to a 27-point stencil.

All this information is computed at the start of the run and saved. Presently, since we do not have a regridding routine, the topological description of the grid does not change during the course of a run—even in a Lagrangian calculation.

EOS Package

In the discontinuous finite element method, density, pressure, and velocity are linear functions. These variables are used to calculate the hydrodynamic fluxes. The Roe solver (which computes the fluxes) needs various derivatives of thermodynamic quantities. This requires that the EOS module compute some variables as functions of different combinations of other quantities. In particular, we need

$$\varepsilon(\rho, p),\ p(\rho, \varepsilon),\ \frac{\partial\varepsilon(\rho, p)}{\partial p},\ \frac{\partial\varepsilon(\rho, p)}{\partial\rho},\ \text{and}\, \frac{\partial\varepsilon(\rho, T)}{\partial T}\quad .$$

Such functions are part of our C++ EOS class. Compartmentalizing these functions into a class allows for better management of data common to all functions for a specific material. For example, in the case of tabulated equations of state, the functions are calculated from the same database. The ICF3D EOS class uses the C++ concepts of inheritance and virtual function overloading to allow new equations of state to be easily implemented.

Currently, three different types of equations of state are supported by the EOS class: ideal gases, ASCII versions of EOS tables for various materials, and a version of the SESAME equation-of-state tables from the Los Alamos National Laboratory.[7] Tabular data are fitted by bicubic splines. The EOS class "constructor," called after the problem initializes, computes spline coefficients. The coefficients are saved and used to evaluate the functions. This makes for faster calls, at the cost of storing a larger amount of data. We intend to add a class that uses the TABLib library.[8] This will allow ICF3D to use tabular equations of state stored as PDB files.

If a cell has more than one material, ICF3D calculates and advects mass fractions. In this case, the cell's thermodynamic quantities are specified by a total ρ, total energy density (or total *p*, or total *T*), and a set of mass fractions for the different materials.

Hydrodynamics

The ICF3D hydrodynamic scheme is described in Ref. 4. The algorithm advances two scalar hyperbolic equations for the density ρ and the total energy density ρ*E*, and the vector equation for the momentum density ρ**v**. The scheme is compact and is easily parallelized since it gets all of its accuracy locally; it does not reach out to more than one neighboring cell to approximate the dependent variables.

The two essential features of the hydro scheme are its ability to do fully Arbitrary Lagrangian Eulerian (ALE) calculations and its ability to run in different coordinate systems. ALE combines the best features of Eulerian and Lagrangian codes. The "arbitrary" aspect allows the user to specify the mesh's motion to resolve a feature that would not be possible to resolve with either a purely Eulerian or a purely Lagrangian code. The hydro module is implemented in 3-D Cartesian, cylindrical, and spherical geometries. We are continuing to develop the ALE features to ensure code robustness. In particular, we have focused attention on two areas. The first involves the Lagrangian limit of the ALE code where the mesh follows the fluid motion. This is a nontrivial problem since discontinuous functions represent the velocity fields, whereas the mesh is required to be continuous. We have developed a scheme to move the grid points that ensures a vanishing average mass flux across a face.

Secondly, the shock stabilization algorithm has been adapted to combine the pure Runge-Kutta solution, the 3-D generalized Van Leer stabilized solution (the limiting procedure), and the first-order Godunov limit solution. This "adaptive" combination assures the greatest possible accuracy as we go from very smooth regions to extremely strong shocks. However, in 2- and 3-D problems, this stabilization is insufficient to stabilize extremely strong shocks. We have implemented a Lapidus-type artificial viscosity[9] to stabilize the remaining "transverse" oscillations near such shocks.

We have successfully run the code on two test problems of particular interest to ICF: the 3-D Rayleigh–Taylor (RT) instability growth problem and the 2-D Richtmyer–Meshkov (RM) shocked perturbation problem. The ressults are described in the "Hydrodynamic Problems" section of this article (see p. 175).

Heat Conduction

Currently, ICF3D uses only a single matter temperature *T*. The change of the internal energy ε due to temperature is expressed as

$$\rho c_v \partial_t T = \qquad \kappa \;\; T + S_\varepsilon \quad , \tag{16}$$

where S_ε is an external source term, κ is the conductivity, and $c_v = \partial_\varepsilon / \partial T$ is the specific heat. In the future, sources such as laser energy deposition will be incorporated into S_ε.[10]

Equation (16) is discretized by a standard nodal FE scheme in which *T* is given a continuous "piecewise linear" representation. The dependent variable is the nodal temperature value. For testing purposes, we provide three conductivity models: κ = constant, a power law, and one given by the Spitzer–Härm formula.[11]

The heat conduction package gathers the various coefficients (c_v, κ, etc.) of Eq. (16) and calls the mathematical diffusion solver package described in "FE Diffusion Package" (see p. 170).

Radiation Transport

Radiation transport is modeled with the diffusion approximation. The coupling to matter is governed by the opacity κ_ν. The relevant equations are

$$\partial_t u_\nu = \qquad D_\nu \;\; u_\nu + c\rho\kappa_\nu \left[B_\nu(T) - u_\nu \right] \tag{17}$$

and

$$\rho c_v \partial_t T = -\int d\nu c \rho \kappa_\nu \left[B_\nu(T) - u_\nu \right] \quad , \tag{18}$$

where $B_\nu(T)$ is the Planck function. In the above equations the unknowns are the radiation energy densities u_ν and the matter temperature *T*.

The same FE scheme used for heat conduction discretizes Eq. (17). In addition, u_ν is discretized with respect to frequency: for each range, or frequency "group," u_ν is its average radiation energy density.

To solve the two equations, we use the "fully implicit" scheme described in Ref. 12. First, the source term is linearized about the temperature at the start of the time cycle

$$B_\nu(T) = B_\nu \Big|_{T_o} + dB_\nu / dT \Big|_{T_o} \left(T - T_0 \right) \quad . \tag{19}$$

The discretization leads to a large system of linear equations, which is solved by a matrix-splitting iteration. We write the system as $A\mathbf{x} = S$, where $\mathbf{x} = (u_\nu, T)$ denotes the vector of unknown energy densities and temperature. As described below, we split *A* in two ways: first, as $A = A_d + A_c$, then, as $A = A_l + A_{nl}$. For the first splitting, we solve

$$A_d\mathbf{x}_1 = S - A_c\mathbf{x}_0 \quad .$$

For the second, we solve

$$A_l\mathbf{x}_2 = S - A_{nl}\mathbf{x}_1 \quad .$$

The two splittings constitute one iteration. The iterations are repeated until u_ν and T converge.

In the first splitting, A_d is the part of the matrix that arises from the diffusion operator in Eq. (17). A_c is the remaining part of A and includes the radiation–matter coupling. The first splitting is equivalent to solving for u_ν while keeping T fixed. Since the groups are only coupled through T, the individual group densities may be solved independently of each other. This results in repeated calls to the diffusion solver.

In the second splitting, A_l is the "local" part of the matrix, formed from A by ignoring all spatial coupling. A_l is a block diagonal matrix, with each block corresponding to a single mesh point. The blocks are of order $N_\nu + 1$. By ordering u_ν first, it follows that each block consists of an order N_ν diagonal matrix in the upper left corner and completely filled last row and column. Hence, the blocks are quickly solved with Gaussian elimination.

We plan to experiment with other linear solution methods and to choose the ones that offer the best combinations of speed and accuracy. One such method is the "partial temperature" scheme.[6] This method has the advantage of not requiring any iterations. For the price of advancing a single diffusion equation once, one gets u_ν and T implicitly coupled.

We also plan to add the advection of u_ν. The above method is physically correct only when the code runs in the Lagrangian mode. For Eulerian or other modes in which the mesh does not move with the fluid, we need to consider that the radiation is also carried by the fluid.

Lastly, we need to incorporate a routine to calculate realistic opacities κ_ν. Presently, ICF3D has only three types of opacities: a user-specified constant, a simple formula $\kappa_\nu = (T/\nu^3)(1 - \alpha e^{-\nu/T})$, and tabulated "cold" opacities.

FE Diffusion Package

The heat conduction and radiation diffusion modules both call the diffusion package, a routine that solves the generic equation

$$g\partial_t f = \quad D \quad f + S - \alpha f \quad . \tag{20}$$

The diffusion package transforms Eq. (20) into a linear system and calls the solver. Equation (20) is discretized by FE methods. The unknown function f is represented in terms of basis functions

$$f = \sum \varphi_j(x) f_j \quad ,$$

where φ_j is the usual piecewise linear function $\varphi_j(x_i) = \delta_{ij}$. The main difference between the diffusion basis functions and those used for the hydrodynamic module is that for diffusion, support (φ_j) extends over all cells with x_j as a vertex.

The time derivative in Eq. (20) is discretized using fully implicit differencing. With one exception, all coefficients are assumed to be known and constant within a cell. The exception is the radiative source, Eq. (19), in which T and T_0 have nodal representations.

After discretizing the temporal derivative, Eq. (20) is multiplied by $\varphi_i\,\Delta t$ and integrated over the entire domain. This leads to a sparse linear system for the coefficients f_j,

$$\left(g + \alpha - \quad D \quad\right) f = g \quad f_0 + S \quad . \tag{21}$$

Integration by parts turns the transport term into a surface integral and a volume integral of the type

$$\int_\Omega dV \; D \quad \varphi_i \quad {}_j\varphi \quad . \tag{22}$$

If x_i is not a boundary node, or if one prescribes a boundary symmetry condition for the flux ($-D\partial f/\partial n = 0$), then the surface integral does not appear. All terms except the transport term are lumped. This implies that only the flux contributes to the off-diagonal coefficients of the matrix. Because of the symmetry in Eq. (22), the matrix is symmetric. It is easy to show that the matrix is also positive definite.

Unfortunately, we may not get an M matrix since some off-diagonal coefficients may be positive. The M-matrix property—only non-negative coefficients for the matrix inverse—is desirable since the diffusion module advances positive quantities such as T. The right-hand side of Eq. (21) is itself positive since it is a sum of the source and the old energy. To show the loss of the M-matrix property, recall that the transport term is discretized by Eq. (22). The diagonal contribution with $i = j$ is clearly positive. For the off-diagonal terms, consider the contribution to Eq. (22) from just one element, K. By construction, D is constant and positive over the element. Hence, let $D = 1$. In 2-D triangles, it is easy to show that $\quad \varphi_i \quad \varphi_j \le 0$, if none of the interior angles are obtuse. In 2-D quadrilaterals, the result is more restrictive, since the sign depends on the aspect ratio of the sides. The issue is exacerbated in 3-D. Consider the hexagon

$$K = \left\{(x, y, z) : |x| \le 1\, , |y| \le Y,\; |z| \le Z\right\}$$

and the functions

$$\varphi_\pm = (1 \pm x)(Y + y)(Z + z)/8YZ \quad .$$

A direct integration yields

$$\int_K dV \quad \varphi_+ \quad \varphi_- = \left(Y^2 + Z^2 - 2Y^2Z^2\right)/9YZ \quad . \tag{23}$$

Again, the sign depends on the aspect ratio. If $Y = Z$ and $Y < 1$, we get the undesired positive sign.

It remains to be seen whether the lack of the M-matrix property proves harmful. It does make the system harder to solve (see the section below, "Solution of Linear Systems.") We are aware that for the case: $S = \alpha = 0$ and $g = 1$, the linear system of Eq. (21) has the form

$$(M + \Delta t S)f = Mf_0 \quad , \tag{24}$$

where M is the lumped (diagonal) mass matrix and S, the stress matrix, is the discretization of the diffusion operator. For negligibly small Δt, $(M + \Delta t\ S)^{-1} \approx M - \Delta t S$. Hence, if S has positive off-diagonal terms, then for a properly chosen and still positive f_0, we may obtain an f with some components negative.

Once the diffusion module has initialized the linear system, it calls the linear solver.

Solution of Linear Systems

ICF3D generates two types of linear systems. One kind arises when the hydrodynamic module inverts mass matrices to compute the fundamental variables. The order of those systems equals the number of degrees of freedom in a cell, e.g., eight in a hexahedron. To solve these systems, we have written a set of general routines that perform Gaussian elimination with partial pivoting.[13]

The second type of systems is created by the diffusion module. For such problems, the matrix A is large, sparse, symmetric, and positive definite (SPD). Because of the unstructured grid, the matrix sparsity pattern is random. SPD systems are best solved by the preconditioned conjugate gradient (PCG) method. Three preconditioners are available: Incomplete Cholesky (IC), one-step Jacobi (diagonal scaling), and multistep Jacobi. In this section, we only discuss the uniprocessor version of the solver. "Parallel Solution of Linear Systems" (see p. 175) describes the MPP modifications.

The PCG algorithm is well documented in the literature.[13] We shall not rederive it here, but instead describe our implementation. Unless the grid topology is changed, the sparsity pattern remains the same. Hence, if the discretization couples node x_i to node x_j, then the i^{th} row of the matrix has a nonzero entry in the j^{th} column. Since the matrix is symmetric, row j has the same entry in the i^{th} column. Thus, the following data structure suffices to describe the sparsity pattern. For each row i, we define an integer array C_i of dimension $C_{i,\,dim}$ where the integer $C_{i,\,dim}$ is the number of nonzero entries in row i to the left of the diagonal. For some rows, for example the first, $C_{i,\,dim} = 0$. The array C_i contains the column numbers of the nonzero matrix entries. For example, if the seventh unknown is only coupled to the third and fifth, $C_{7,\,dim} = 2$ and $C_7 = (3, 5)$.

This description is computed only once, at the start of the run. The pattern is computed from the nodal neighbor construct described in "ICF3D Initialization" (see p. 168). Of course, if the problem were to be regridded (and the mesh topology changed), the sparsity pattern would be recomputed. Although the above discussion focuses on a sparsity pattern due to an FE discretization of a nodal quantity, the procedure can be generalized. For example, if we need to discretize a zonal quantity, we first determine the zone's neighbors for this application and then follow the same procedure. The linear solver need not distinguish how the linear system was derived. All we supply is a description of the system, i.e., the matrix order and sparsity pattern, the matrix elements, the maximum number of iterations allowed, the preconditioner desired, etc.

To use the IC preconditioner, we do more preliminary work. For uniprocessors, the IC variant factors the matrix into the product

$$A = LDL^T \quad ,$$

where D is diagonal, L is lower triangular with unit diagonal and a sparsity pattern that matches that of A, i.e., no fill-in. It can be shown[14] that computing an entry in the lower triangle of L, e.g., row i and column j, involves a dot product of the previously computed entries of row i and row j. For efficiency, we do not compute products of a nonzero entry in one row by a zero entry in another. Avoiding such unnecessary multiplications requires extra preliminary work and additional storage in order to describe which entries contribute to the product. As a result, the evaluation of L proceeds faster, but with the penalty of indirect addressing of the necessary elements.

We have only tested our solver on small problems. First of all, we have confirmed that for sparsity patterns in which all nonzero matrix entries are bunched about the diagonal, IC returns the exact decomposition and PCG converges in one step. Secondly, we obtain the results in Table 1 for Laplace's equation on the unit cube with Dirichlet boundary conditions. Using a uniform grid with $L = 1/(n + 1)$ and $n = 15$, we obtain a system of order n^3.

Note that even though two-step Jacobi took nearly 50% more iterations than IC, the time spent was almost

TABLE 1. Results for Laplace's equation on the unit cube with Dirichlet boundary conditions $L = 1/(n+1)$ and $n = 15$. Time is in arbitrary units.

Preconditioner	Iterations	Time
One-step Jacobi	30	—
Two-step Jacobi	18	56.15
Incomplete Cholesky	13	56.13

identical. The nearly twofold decrease in the iteration count between one- and two-step Jacobi agrees with the results of Ref. 15. However, this problem is characterized by uniform and equal Δ_x, Δ_y, and Δ_z, which implies that we obtain an M matrix. Hence, the first-order Jacobi iterative method converges,[16] and one can apply the results of Ref. 15 to make the two-step Jacobi method an efficient preconditioner.

However, for nonuniform meshes, Eq. (23) implies a loss of the M-matrix property. Furthermore, it is easily shown that the matrix need not be strictly diagonally dominant. Hence, the Jacobi method need not converge and the two-step preconditioner may even delay convergence. To illustrate, we consider the same problem as above except we set $X_{\max} = 1$ and $Y_{\max} = Z_{\max} = 0.1$. We again use $n + 1$ uniformly spaced points in each direction. The results appear in Table 2.

TABLE 2. Same problem as for Table 1, with $X_{\max} = 1$, and $Y_{\max} = Z_{\max} = 0.1$.

n	Preconditioner	Iterations
11	None	40
11	One-step Jacobi	37
11	Two-step Jacobi	40
11	Incomplete Cholesky	10
21	None	58
21	One-step Jacobi	56
21	Two-step Jacobi	441

For $n = 11$, two-step Jacobi is slightly worse than one-step and no better than the conjugate gradient method without a preconditioner, while IC took four times fewer iterations. However, for $n = 21$, two-step Jacobi actually did considerably more harm than good.

Lastly, in the above problem, we confirmed that the numerical solution equals the exact solution, $(1 - x)(1 - y)(1 - z)$. On a hexagonal mesh, the exact solution belongs to the domain spanned by the test functions.

Problem Generation

The task of generating truly unstructured meshes, even those limited to hexagonal, prismatic, pyramidal, and/or tetrahedral cells, is a formidable job and is, in itself, the subject of ongoing research of other groups and companies. For our test problems, we improvised by writing a separate code, the generator, that creates only logically hexagonal grids.

To each point, the generator assigns an integer three-tuple (k, l, m), where the indices are positive and bounded by $(k_{\max}, l_{\max}, m_{\max})$. The generator also specifies the initial and (currently, time independent) boundary conditions, the material(s) of choice via EOS tables, and discretizes the frequency spectrum. To facilitate the generation, we have enveloped this process under a controller or script language parser. The system was originally developed under Basis,[17] but we have adopted Python,[5] a new, portable language that is easily extended. Using a technique similar to that for LASNEX, a set of generator functions describe the problem in an ASCII file written in Python. After the generator reads the file, it writes the ICF3D input files in the UCD format.

The generator has been extended to specify meshes that are not logically hexagonal. We allow for voids within the domain, e.g., to represent solid bodies. The generator also allows the user to describe the grid in one coordinate system and write the input file in another. Thus, we may discretize the sphere in (r, θ, φ) and compute in Cartesian coordinates. This presents a special challenge, because the individual elements are not symmetric in the expected directions. The generator also decomposes the domain for MPPs. The user chooses along which of the k, l, and/or m directions to parallelize. For example, one could divide a $k_{\max}$, $l_{\max}$, $m_{\max}$ = (31, 51, 81) domain amongst 2 × 2 × 4 = 16 PEs, where each PE owns 15 × 25 × 20 cells.

Python can interact with and "steer" the execution. Steering permits the user to directly interact with the various code modules, thereby allowing real-time analysis of the computed results. Unfortunately, this mode of operation requires that the controller/interpreter be portable to the machine of choice. This requirement may not be simple to fulfill, particularly on MPPs. Consequently, we provide two modes of operation. In one, which is still under development, the computing core (ICF3D) and the controller (Python) are tightly coupled. We use this mode on uniprocessors. The second mode gives us the flexibility of running ICF3D on a variety of platforms with no concern of the steering engine's portability. This mode evokes the distributed computing model; the stand-alone ICF3D is controlled by a Python session executing at a local workstation.

ICF3D is run in this second, distributed mode, on a variety of computers, uniprocessors and MPPs. This allows us to use the parallel machines exclusively for computations and relegates the generation and postprocessing chores to the desktop. The user directs where to execute ICF3D, and the generator takes care of the details: reading the user's deck, preparing the input files, shipping them to the computer of choice, initializing the run, etc. While this is not as flexible as a truly steerable code where parser and code are intimately connected in a single executable, it has given us an invaluable means to develop and debug ICF3D and has provided an easy interface to the MPP of choice. The system supports continuation runs via restart files as well as a link to the debugger on the remote machine. Our only requirements on the remote machine are a correct C++ environment with standard remote-shell execution capabilities. Since the interface is the same regardless of the computing engine—MPP or uniprocessor—the parallelization is completely transparent to the user. The decomposition of the domain and subsequent recombination of the results from the individual PEs are done by the generator. In a typical run, after ICF3D finishes, the data is automatically transmitted to the workstation to be saved for later postprocessing or examined immediately. For subsequent processing with the AVS visualization software, ICF3D also writes files in the AVS UCD data format.

Parallel Processing

We target distributed memory processors (DMPs) and rely on explicit calls to message passing functions. This is the optimal strategy for computers such as the CRAY T3D and the IBM SP2, as well as networks of workstations (if the network is provided with the appropriate message-passing library). The strategy also works on shared memory computers (SMPs). ICF3D has successfully run on uniprocessor workstations and DMPs such as the T3D, as well as on SMPs like the 12-node SGI Power Challenge.

We divide the physical domain into nearly disjoint subdomains, one per PE, and "leave" the subdomains distributed on the PEs. Each PE receives a description of only its subdomain plus a layer of adjacent ghost cells. In Fig. 1, we display the entire subdomain (including ghost cells) sent to one PE. The subdomain consists of 96 owned cells and 236 ghosts. (This example has a bad surface-to-volume ratio.) There are 108 hexahedra, 96 prisms, 96 pyramids, and 32 tetrahedra. Only 48 hexahedra, 16 prisms, 24 pyramids, and 8 tetrahedra are owned cells. The peculiar cones and innermost sphere primarily consist of ghost cells. They appear since the domain decomposition requires that each owned cell know about all cells that share one of its vertices.

The subdomains are described in the input files, which are written in the modified AVS UCD format. The modification consists of assigning global as well as local sequence numbers to both cells and nodes, and tagging the cells with the number of the PE that owns it. The input files are written by the generator; and only it has access to the entire domain. The generator initializes the problem, decomposes the domain, and determines which PE owns which cell. The assignment of nodes to specific PEs is done by ICF3D itself. Thus, since cells do not migrate across the PEs, the task of load balancing the PEs is assigned to the generator.

We use a "coarse grained" parallelization strategy; the code is not replete with parallelization commands. Crucial code segments have been identified as requiring information that resides on another PE. At such segments we insert statements of the type

```
if (Number PEs >1) call fooMPP();
```

Such branching statements appear at a high level in the code.

For parallelization purposes, the physics modules may be roughly divided into three kinds: embarrassingly parallel, clearly parallelizable, and hard to parallelize efficiently. The first type consists of modules such as the EOS; nothing special is done since each PE works only on the cells it owns. Explicit time-differencing methods such as the hydrodynamic step are clearly parallelizable. Such methods have an easily identifiable, compact domain of dependence. For example, a cell's hydro variables at one time step depend on only

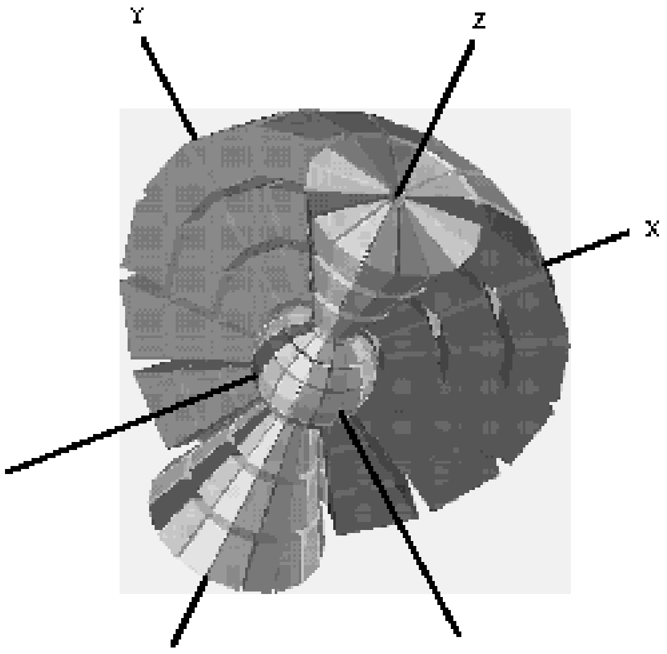

FIGURE 1. Entire subdomain given to a PE when decomposing a sphere and running in Cartesian coordinates. Figure includes both owned and ghost cells. (50-06-1296-2785pb01)

old values of that cell and its neighbors. If the neighbor cells are owned by the same PE, no extra work is required. If some of the surrounding cells belong to another PE, a copy of that cell is a ghost cell of this PE. It is the responsibility of the message-passing routines to get the required information. The third, hard-to-parallelize-efficiently are those modules that require global communication, e.g., the initialization and solution of linear systems. That subject is discussed in "Parallel Solution of Linear Systems" (see p. 175).

When running in parallel, we use one of two libraries in one of two modes. In one mode, for portability, we call routines from the industry standard MPI message-passing library. On the Cray T3D, we also provide the option of using the native SHMEM library, since efficient implementation of MPI is site-dependent. In the early stages of our parallelization work, the LLNL MPI library was extremely slow. Now, the SHMEM and MPI performances are nearly equivalent. Nevertheless, having a choice of libraries has proved invaluable; when one breaks, we switch to the other. In the other mode, we have an additional object-oriented layer, C4,[18] which is itself written in C++ and thus allows better use of that language. For example, MPI routines require that we explicitly state the data type of the arguments. In C++, such type-description is unnecessary since, unlike in FORTRAN, the argument carries information about its type. C4 has also been generalized so that it either calls SHMEM or MPI.

The interface to the MPI, SHMEM, or C4 functions is through special message-passing objects (MPOs), which are constructed at the start of the run. These objects store the information required to effect the message passing. For example, assume that at each time step, PE_0 sends to PE_9 nodal data corresponding to nodes with global sequence numbers 500 and 1,000,001. These nodes will have different local sequence numbers, e.g., 10, 11 on PE_0, and 50, 70 on PE_9. The MPO stores the number of the other PE, the length of the message, and the local sequence numbers on this PE. Thus, on PE_9 the MPO knows it is to receive two numbers from PE_0 which are to be stored in locations 50, 70.

Whenever we enter a module that requires communication, such as the hydro package, we allocate buffers of the proper size to store the messages. The buffers are deallocated when that module is exited. In the future, to avoid repeated calls to memory allocation routines, we will allocate a permanent block of memory during the initialization step and have the MPO reuse this block.

Parallel Hydrodynamics

The hydrodynamic module is fully parallelized. We use three kinds of MPOs. (Only two are needed for Eulerian calculations.) One communicates facial information, the other nodal. The former is used to compute fluxes (on cell faces) on inter-PE boundaries. The latter is used in the limiting process, which deletes unphysical extrema in the solution. For Lagrangian runs, additional MPOs pass the required information to compute the velocity of nodes lying on inter-PE boundaries. We have verified that, to round-off precision, the results are independent of the partitioning of the domain amongst the PEs.

The hydro scheme, being explicit, lends itself nicely to parallelization, with results scaling with the number of PEs used. In Table 3, we display timings from a test problem. The problem is run in Cartesian coordinates, but the mesh is generated with cylindrical coordinates. In the results, we fix the number of cells and increase the number of PEs. Let N_{PE} denote the total number of PEs, PE_{config} the PE configuration in the (R, θ, Z) directions, $cell_{config}$ the number of owned cells in (R, θ, Z), $cell_{ratio}$ the ratio of owned to ghost cells, Time the execution time, Speedup the ratio of Time to Time for the 32-PE run, and E = Speedup/(N_{PE}/32) the efficiency. The efficiency results reflect the fact that as N_{PE} increases, $cell_{ratio}$ degrades, which creates a greater message passing overhead.

TABLE 3. Timing results (in s) obtained from a test problem run in Cartesian coordinates and with mesh generated with cylindrical coordinates.

N_{PE}	PE_{config}	$Cell_{config}$	$Cell_{ratio}$	Time	Speedup	E
32	(8, 1, 4)	(8, 16, 8)	1.32	959.7	—	—
64	(8, 1, 8)	(8, 16, 4)	0.90	501.0	1.92	0.96
128	(16, 1, 8)	(4, 16, 4)	0.74	264.6	3.63	0.91
256	(16, 1, 16)	(4, 16, 2)	0.42	149.9	6.40	0.80

Parallel Solution of Linear Systems

At the time of writing this article, we are still developing parallel solutions to linear systems. What follows here is a general discussion of the topic; timings and scalings will be presented at a later date.

The linear systems generated by ICF3D are large, sparse, symmetric, and positive definite, and therefore ideally suited for PCG. For uniprocessors, we offer three kinds of preconditioners: IC, one-step, and multistep Jacobi. Their parallel implementation is described below.

The CG iterations consist of vector sums, inner products, matrix vector multiplications, and the solution of the preconditioning system. These operations are easy to implement if the nodes are assigned to the PEs. This is done in parallel at the start of the run. Nodes on or inside inter-PE boundaries are assigned without any need for message passing. Ownership of nodes on the "outside" of the ghost cells requires point-to-point communication. After this initialization, every node on every PE has been tagged with the PE that "owns it."

Once the nodes have been assigned, the vector sums are trivial to implement. Each PE only updates the nodes it owns. For the inner products, each PE computes a preliminary accumulation of its owned nodes and then calls a global "reduction" routine provided by the message-passing library. These calls are potential bottlenecks since they require an all-to-all PE communication.

The matrix is distributed amongst the PEs as follows. We use a row-wise assignment of the matrix coefficients. Each PE eventually gets all the matrix coefficients for the rows corresponding to nodes that it owns. The coefficients come from integrations over cells, e.g., Eq. (22). If x_i is owned by the PE but lies on an inter-PE boundary, the contribution from a particular cell is done by the PE that owns the cell. With the FE scheme, each PE computes the matrix coefficients just as in the uniprocessor case, i.e., it integrates only over owned cells. Then, special MPOs determine which contributions need to be exchanged with other PEs. As for the hydro MPOs, the matrix MPOs are constructed during the initialization phase.

Once the matrix is distributed, the matrix vector multiplications are relatively straightforward. If

$$y_i = \sum_j A_{i,j} x_j \tag{25}$$

is one of the owned elements of the PE, the product is easily done if the PE has the latest x_j values. For some y_i, the requisite x_j are owned by other PEs. Those values are delivered to the PE by other MPOs.

Lastly, we discuss the preconditioners. For one-step Jacobi, message passing is not required. Multistep Jacobi, which is one example of a polynomial preconditioner,[13] requires a matrix vector multiplication. Each step is preceded by a send/receive. For IC, our plan is to perform the usual ICCG(0) algorithm but ignore coupling across the PEs. In this way, we avoid the fundamental bottleneck of ICCG, namely, how to parallelize the solutions of the triangular systems. If the domain decomposition has a low surface-to-volume ratio, this approach is efficient. Effectively, the decomposition is even more incomplete than on a uniprocessor. At the very least, this parallelizes, does not require any message passing, and should outperform one-step Jacobi because the latter is the limiting case of a decomposition so incomplete that it ignores all coupling.

Numerical Results

In this section we present results of test problems in order to display some of ICF3D's capabilities and check the algorithms. The section is divided into three parts. "Hydrodynamic Problems" (see below) discusses two hydrodynamic problems—the Rayleigh–Taylor and Richtmyer–Meshkov instabilities—which use only the hydro physics module and ideal gases, i.e.,

$$(\gamma - 1)c_v T = (\gamma - 1)\varepsilon = p/\rho \quad ,$$

where γ and c_v are user-specified constants. In "Non-linear Diffusion Problem" (see p. 177), we turn our attention to the heat conduction module and simulate nonlinear diffusion. The calculation is done in 3-D Cartesian coordinates, but by construction, the problem should be spherically symmetric. Finally, in "Coupled Physics Problem" (see p. 178), we discuss a contrived test problem that exercises all of the available physics modules, a realistic material, hydro, heat conduction, and radiation, and compare our results to LASNEX.

Hydrodynamic Problems

We consider a class of instabilities where two fluids of different densities are subjected to an acceleration $\mathbf{g}$. Theoretical work on the growth of a perturbed interface in which a light fluid supports a heavy fluid was originally presented by Taylor[19] who studied perturbations of the type

$$\delta z = a \cos kx \quad , \tag{26}$$

where z_0 = const., $ka << 1$, and the wave number $k = 2\pi/\lambda$. Taylor showed that the amplitude a satisfies

$$\frac{d^2 a}{dt^2} = kg(t)a(t)A \quad , \tag{27}$$

where $A = (\rho_2 - \rho_1)/(\rho_2 + \rho_1)$ is the Atwood number, defined in terms of the densities on either side of z_0, and where $\mathbf{g} = -g\hat{\mathbf{z}}$.

We simulate two cases: (1) the Rayleigh–Taylor instability in which g = const. and (2) the Richtmyer–Meshkov instability in the acceleration is impulsive, e.g., caused by a shock.

Rayleigh–Taylor Instability

We consider the nearly incompressible case suggested by Tabak;[20] we set $\gamma = 10$ in both gases. The computational domain consists of a box with axial extent, $0 \le z \le 2\lambda$. The initial equilibrium density distribution is

$$\rho = \begin{cases} 10(z/l)^{1/9} & 0 \le z \le \lambda \\ 100(z/l)^{1/9} & \lambda \le z \le 2\lambda \end{cases} \tag{28}$$

where $l = 1.1\lambda$. We run this problem in the Lagrangian mode. At $t = 0$, we initialize the grid with a sinusoidal perturbation at $z = \lambda$ with amplitude $a = 2.25 \times 10^{-4}\ \Delta z$ where $\Delta z = 2\lambda / N_z$ is the unperturbed grid spacing. The perturbation is feathered into the z direction three zones deep on either side of the interface. We consider three problems. In each one, we initialize one-half wavelength in the transverse direction. In all cases we use 40 cells in the z direction. The problems are

- A 2-D case with 20 cells in the x direction. The perturbation is given by Eq. (26) with $z_0 = \lambda$.
- A 3-D case with 20 cells in both x and y directions. The perturbation is given by

$$\delta z = z_o = a\cos\left(k_x x\right)\cos\left(k_y y\right) \quad , \text{ and } k_x = k_y .$$

- A 3-D case with 20 cells in both x and y directions. The perturbation is as above except $k_x = 3k_y$.

The wavelengths are chosen so that all cases have the same total wave number $k = \sqrt{k_x^2 + k_y^2}$.

For constant g, Eq. (27) implies that the perturbation grows exponentially with growth rate $\Gamma = \sqrt{Agk}$. Hence, all three cases should grow at the same rate. With $k = 2\pi/0.001$ and $g = 0.33671717$, we obtain $\Gamma = 41.6$. Figure 2 displays $\log_{10}(a)$ vs t for $t \le 0.25$ and $t \ge 0.05$, after the system has settled into an eigenmode. After 0.25, the mode reaches an amplitude of approximately 0.1 λ and saturates. The numerical Γ is within 1% of the theoretical value. This test problem shows the ability of ICF3D to model the linear regime thereby demonstrating the accuracy of the method. In Fig. 3, we display the interface at $t = 0.14$ for the $k_x = k_y$ case.

Richtmyer–Meshkov Instability

In this problem, analyzed by Richtmyer,[21] a perturbed interface interacts with an incident planar shock. Before the shock's arrival, the interface is in equilibrium since the two gases have equal pressures. From 1-D theory (no perturbation), a shock-on-contact interaction results in a displacement of the interface and two waves, one transmitted, another reflected. The transmitted wave is always a shock. The type of reflected wave depends on the unshocked density of the gas on the side from which the shock is traveling. If this side is of higher (lower) density than the other gas, the transmitted wave is a rarefaction (shock). If the "shock-incident" side is of higher density, the amplitude of the perturbation changes sign. In either case, after the interaction, the amplitude is abruptly diminished, but later grows linearly in time.

For the simulation, we run in the Eulerian mode to avoid tangling the mesh and to follow the growth into the nonlinear regime. In the results, z is the horizontal direction and x the vertical. We use the specifications relayed by Dimonte.[22] The domain consists of $(0, 0) \le (z, x) \le (0.04, 0.005)$. We simulate only half a wavelength along x and apply symmetry conditions at transverse boundaries. For z, we apply symmetry at $z = 0$, and inflow at $z = 0.04$. The initial perturbation is given by Eq. (26) with $a = 10^{-3}$, $k = 200\ \pi$, and $z_0 = 0.03$. The initial conditions are characterized by three regions: the left $z < z_0$, the right unshocked $z_0 \le z < z_s$, and the right shocked $z_s \le z$, where $z_s = 0.32$ is the position of the incident, planar shock. The conditions are described in cgs units in Table 4.

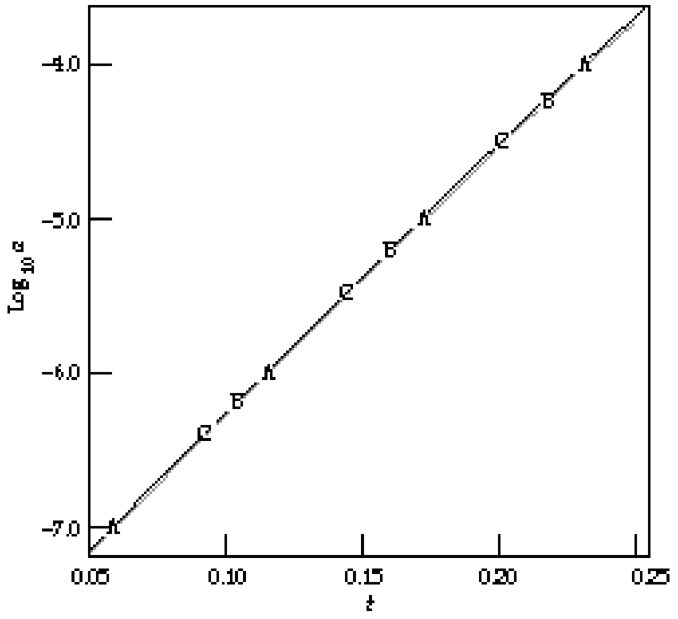

FIGURE 2. Rayleigh–Taylor problem showing common logarithm of mode amplitude vs time. Labels A, B, and C correspond to 2-D, 3-D square, and 3-D asymmetric cases, respectively. (50-06-1296-2786pb01)

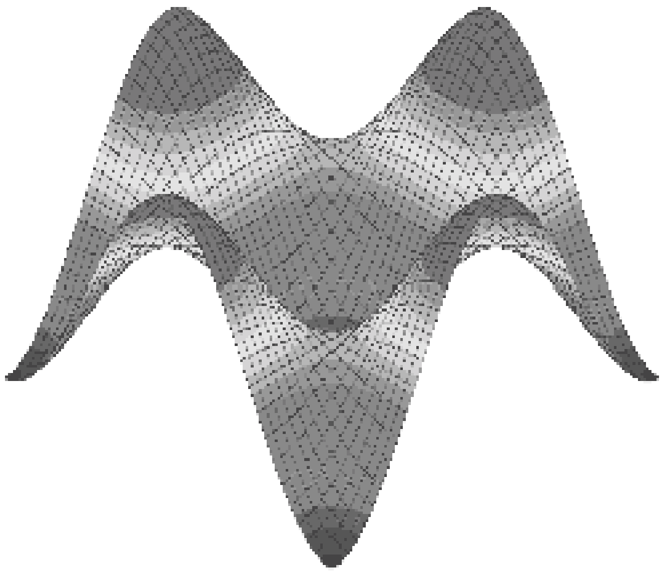
FIGURE 3. Rayleigh–Taylor problem showing perturbed interface at $t = 0.14$ ns. (50-06-1296-2787pb01)

TABLE 4. Conditions (cgs units) used in the Richtmyer–Meshkov instability problem.

Region	Density	Velocity	Pressure	γ
$z < z_0$	0.12	0	$4.92048\ 10^{10}$	1.45
$z_0 \le z < z_s$	1.7	0	$4.92048\ 10^{10}$	1.8
$z < z_0$	3.6617120	$-2.453647\ 10^5$	$2.4\ 10^{11}$	1.8

With these conditions, we expect a reversal of the interface, followed by a growth of the amplitude and an eventual nonlinear stage setting in. In Fig. 4, we present a time sequence of ρ. Fig. 4(a) shows the initial condition. In Fig. 4(b), $t = 7$ ns, the shock has already hit the interface, producing a transmitted shock and a reflected rarefaction. Note the reversal of the interface. The results in Figs. 4(c) and 4(d), at $t = 22$ ns and 37 ns respectively, show the onset of the nonlinear stage. In Fig. 4(e), $t = 53$ ns, the interface is beginning to form spikes. By this time, the shock has reflected off the left boundary but has not yet interacted with the interface. By the end of the simulation, $t = 77.5$ ns Fig. 4(f), the reflected shock has passed through the interface. The interface is now severely deformed. At this time the shock is at $z = 0.014$ and has itself developed a slight deformation. These results are in qualitative agreement with the ones published by Cloutman et al.[23]

Nonlinear Diffusion Problem

We now consider the spherically symmetric, nonlinear diffusion equation

$$\frac{\partial T}{\partial t} = \frac{a}{r^2}\frac{\partial}{\partial r}\left(r^2 T^n \frac{\partial T}{\partial r}\right) \quad , \tag{29}$$

where a and n are constants. If the initial condition is a delta function and the domain extends to infinity, the analytic solution is given in Ref. 24. By conservation of energy (heat), the solution satisfies

$$\int_0^\infty 4\pi r^2 T dr = Q \quad .$$

The analytic solution is given in terms of the radial position of the front r_f and the temperature at the center T_c in

$$T = T_c\left(1 - r^2 / r_f^2\right)^{1/n} , r \le r_f \quad , \tag{30}$$

$$r_f = \xi\left(aQ^n t\right)^{1/(3n+2)} \quad , \tag{31}$$

and

$$T_c = \left[\frac{n\xi^2}{2(3n+2)}\right]^{1/n}\left[\frac{Q^2}{(at)^3}\right]^{1/(3n+2)} \quad . \tag{32}$$

The constant ξ depends only on the conductivity exponent n. For our test, we use $n = 3$ and obtain

$$\xi\big|_{n=3} = 0.8979$$

For the numerical test, we set $a = Q = 1$ and discretize 1/8 of a sphere: $0 \le r \le 1$ and $0 \le \theta, \varphi \le \pi/2$,

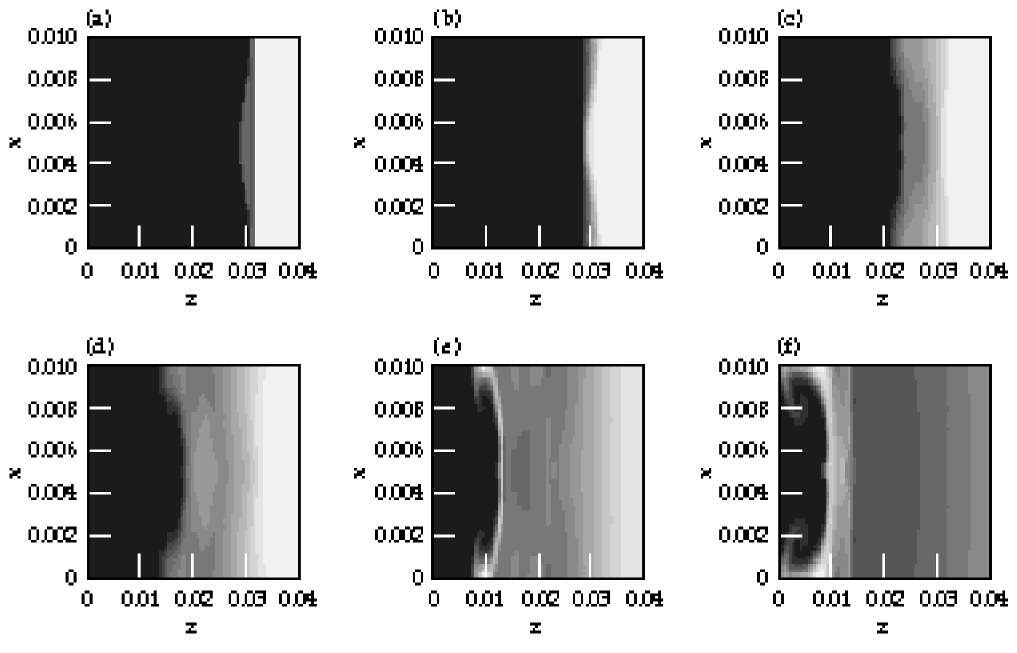

FIGURE 4. Richtmyer–Meshkov problem showing density (g/cm^3) at various times. (a) is at $t = 0$ ns; (b) at $t = 7.0$ ns, (c) at $t = 22.2$ ns, (d) at $t = 36.8$ ns, (e) at $t = 53.3$ ns, and (f) at $t = 77.5$ ns. (50-06-1296-2788pb01)

where θ denotes the polar angle. We impose symmetry conditions along $\varphi = 0$ and $\theta = \varphi = \pi/2$. Along $r = 1$ we also impose symmetry, but this condition is unnecessary since we halt the calculation when $r_f \approx 0.8$.

We generate the grid in spherical coordinates by constructing a logical (k, l, m) grid, where $(1, 1, 1) \leq (k, l, m) \leq (k_{max}, l_{max}, m_{max})$, set $(k_{max}, l_{max}, m_{max}) = (21, 9, 9)$, and use uniform grid spacing. There are $(k_{max} - 1)(l_{max} - 1)(m_{max} - 1) = 1280$ cells. Before running, we convert the grid into Cartesian coordinates. This creates cells of all admissible types since the spherical to Cartesian coordinate transformation results in degenerate nodes. There are $(m_{max} - 1)(k_{max} - 2)$ prisms along the z axis. At the origin there are $m_{max} - 1$ tetrahedra and $(m_{max} - 1)(l_{max} - 1)$ pyramids. The remaining cells are hexahedra.

In Fig. 5 we present the numerical solution at $t = 0.2821$ ns along each Cartesian coordinate axis. The analytic solution given by Eqs. (30) through (32) is also displayed as curve D. The numerical solutions on the three axes are nearly indistinguishable. Examination of the data shows that the central temperature, 0.0671, is within 1% of the analytic value, 0.0676.

Coupled Physics Problem

The final test problem was constructed to test the interaction of all the physics packages presently available in ICF3D: a real gas EOS, hydro motion, Spitzer-Härm heat conduction, and radiation diffusion. The problem models a shock tube filled with beryllium (SESAME table no. 2020); the numbers used in the simulation are not meant to model any experiment.

We run this in a 1-D mode, i.e., only one cell in each of the x and y directions. The initial conditions vary only with z. The initial density ρ is 1.0 gm/cc everywhere. We initialize with a higher (1 keV) temperature on the left, and on the right, we set $T = 0.1$ keV. The initial velocity is set to zero. For the radiation, we use only one group and set the opacity to a constant 1 cm^2/gm. We impose symmetry conditions on the left and right boundaries and run in the Lagrangian mode, using 203 zones in the calculation. Initially, the grid is uniform.

In Fig. 6 we display the pressure at $t = 0.1$ ns when ICF3D computes the joint effect of hydrodynamics, radiation transport, and heat conduction. Since there are no analytic solutions for this problem, we compare the ICF3D calculation to one done with LASNEX. The

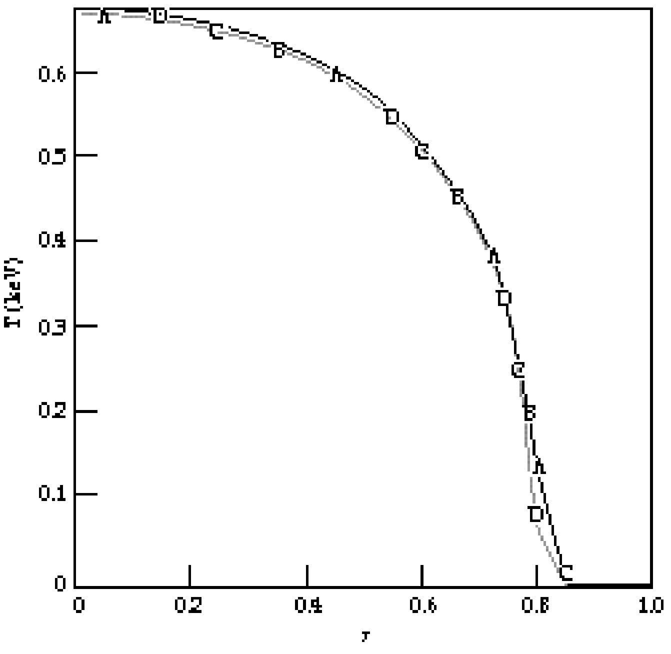

FIGURE 5. Diffusion problem showing temperature profile vs radius along coordinate axes x, y, and z (curves A, B, and C) at $t = 0.2821$ ns. Curve D is the analytic solution. (50-06-1296-2789pb01)

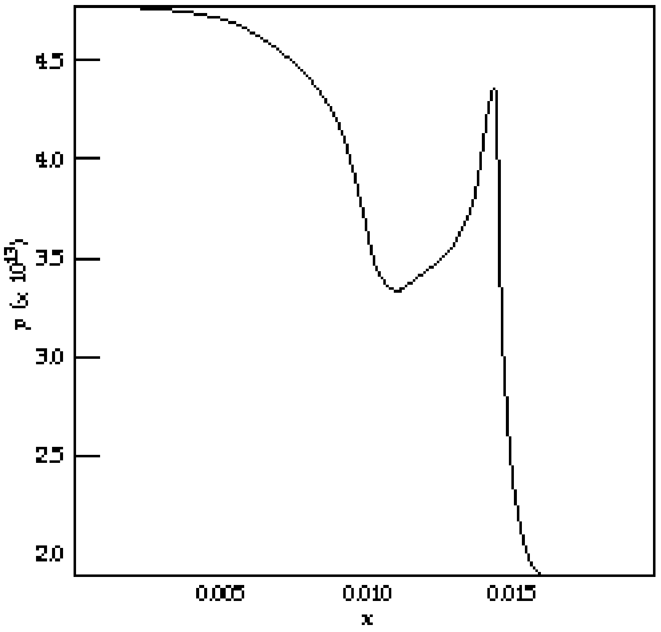

FIGURE 6. Shock tube problem results from ICF3D showing p vs z. All physics running. (50-06-1296-2790pb01)

concern is whether or not the inherently nodal representation of diffusion type problems clashes with the fundamentally cellular representation of the hydro variables. Again, experience in LASNEX as well as the encouraging results from the "Coupled Physics Problem" section (see p. 178) alleviates this concern. Hence, we should be able to reap all of the benefits from FE methods.

So far, the parallelization efforts are a resounding success. In the past year, most of the hydro development effort has been done on the LLNL CRAY T3D because we obtain results that much faster when running on 32 PEs. We are now finishing parallelizing the linear solver and look forward to running coupled problems in parallel.

We have found the C++ programming language to be a helpful tool in organizing a physics code of the complexity of ICF3D. Our experience has given us new ideas on how to organize even larger codes for ease of maintenance and how to program them for maximum speed. We plan to implement these ideas so that ICF3D can grow into the best possible full-scale production code. The full-scale code will also benefit from our research in such areas as numerical hydrodynamics and the use of parallel processors to solve linear equations for diffusion problems.

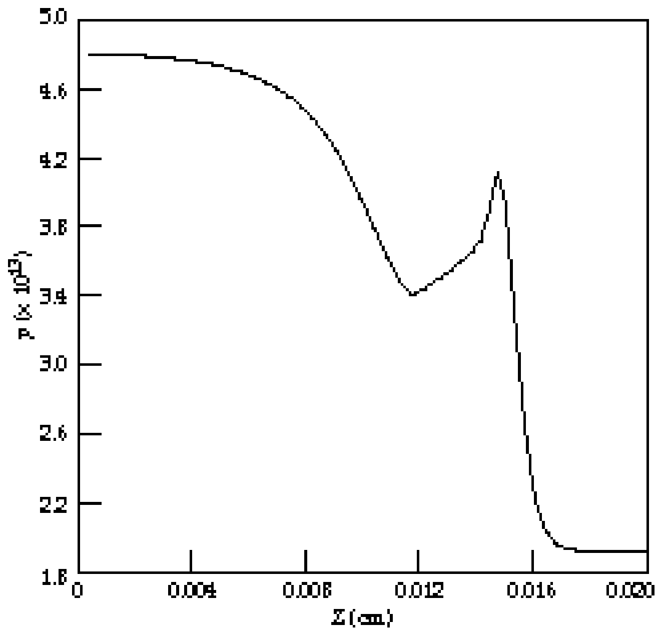

FIGURE 7. Shock tube problem results from LASNEX showing p vs z. (50-06-1296-2791pb01)

Notes and References

1. Furthermore, the time to develop codes does not scale linearly. For example, LASNEX represents a nearly 100 man-year effort; yet 200 developers would not be able to reproduce LASNEX from scratch in six months.
2. S. W. Haney, *Computers in Physics* **8** (6), 690–694 (1994).
3. S. W. Haney, Lawrence Livermore National Laboratory, CA, private lecture series (1996).
4. D. S. Kershaw, M. K. Prasad, and M. J. Shaw, "3D Unstructured Mesh ALE Hydrodynamics with the Upwind Discontinuous Finite Element Method," Lawrence Livermore National Laboratory, Livermore, CA, UCRL-JC-122104 Rev.1 (1996).
5. P. F. Dubois, K. Hinsen, and J. Hugunin, *Computers in Physics* **10** (3), 262-267 (1996). *Python Documentation* [online]. Available: www.python.org.
6. A. I. Shestakov, J. A. Harte, and D. S. Kershaw, *J. Comp. Physics*, **76** (2), 385–413 (1988).
7. "SESAME: The Los Alamos National Laboratory Equation of State Database," Eds. S. P. Lyon and J. D. Johnson, Los Alamos National Laboratory Report LA-UR-92-3407. Also: "T-1 Handbook of Material Properties Data Bases," K. S. Holian, Ed., Los Alamos National Laboratory Report LA-10160-MS November 1984.

8. S. A. Brown, "TABLib User's Manual," Lawrence Livermore National Laboratory internal report (1995).
9. A. Lapidus, *J. Comp. Phys*. **2**, 154–177 (1967).
10. T. B. Kaiser and J. A. Byers, *Bull. Am. Phys. Soc.* **41** (7), 1555 (1996).
11. L. Spitzer, Jr., and R. Härm, *Phys. Rev*. **89** (5), 977–981 (1953).
12. A. Friedman and D. S. Kershaw, *1982 Laser Program Annual Report* , pp. 3–68, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-50021-82 (1982).
13. G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition (John Hopkins Univ. Press, Baltimore, 1989).
14. A. I. Shestakov and D. S. Kershaw, "ICCG on Unstructured Grids," Lawrence Livermore National Laboratory informal report, April 30, 1996. Contact by email: shestakov@llnl.gov.
15. P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, *Computing* **22** (3) 257–268 (1979).
16. R. S. Varga, *Matrix Iterative Analysis* (Prentice Hall, Inc., Englewood Cliffs, N.J., 1962) p. 84.
17. P. F. Dubois, *Computers in Physics* **8** (1), 70–73 (1994).
18. G. Furnish, *Gyrokinetic Simulation of Tokamak Turbulence and Transport in Realistic Geometry*, Ph.D. thesis, Appendix C, Univ. of Texas, Austin, 1996, Institute of Fusion Studies doc. IFSR-740. [Also available online at http://dino.ph.utexas.edu/furnish/c4.]
19. G. Taylor, *Proc. R. Soc. London Ser. A.* **201** (1095), 192–196 (1950).
20. M. Tabak, D. H. Munro, and J. D. Lindl, *Phys. Fluids B* **2** (5), 1007–1014 (1990).
21. R. D. Richtmyer, *Commun. Pure Appl. Math.* **13** (2), 297–319 (1960).
22. G. Dimonte and B. Remington, *Phys. Rev. Lett.* **70** (12), 1806–1809 (1993).
23. L. D. Cloutman and M. F. Wehner, *Phys. Fluids A* **4** (8), 1821–1830 (1992).
24. Ya. B. Zel'dovich and Yu. P. Raizer, *Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena*, Vol. II, Ch. X, Sec. 6 (Academic Press, New York and London, 1967) pp. 668–672.
25. P. Colella and P. W. Woodward, *J. Comp. Phys.* **54** (1), 174–201 (1984).